NCO MegaCore Function

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

MegaCore Version: 8.0
Document Date: May 2008

Copyright © 2008 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

I.S. EN ISO 9001

Part Number: UG-NCOCOMPILER-7.0

Contents

Chapter 1. About This MegaCore Function

Release Information 1–1
Device Family Support 1–1
Features 1–2
General Description 1–3
MegaCore Verification 1–4
Performance and Resource Utilization 1–4
Installation and Licensing 1–6
OpenCore Plus Evaluation 1–6
OpenCore Plus Time-Out Behavior 1–7

Chapter 2. Getting Started

Design Flows 2–1
DSP Builder Flow 2–1
MegaWizard Plug-In Manager Flow 2–2
Generated Files 2–7
Simulate the Design 2–9
Simulating the Design in ModelSim 2–10
Compile the Design and Program a Device 2–10

Chapter 3. Parameter Settings

Setting Parameters 3–1
Parameter Descriptions 3–6

Chapter 4. Functional Description

Numerically Controlled Oscillators 4–1
Spectral Purity 4–1
Maximum Output Frequency 4–2
Functional Description 4–3
Architectures 4–5
Frequency Modulation 4–8
Phase Modulation 4–9
Phase Dithering 4–9
Multi-Channel NCOs 4–10
Timing Diagrams 4–10
Avalon Streaming Interface 4–13
Signals 4–13
References 4–14

Appendix A. Example Designs

Example Design 1 ..... A–1
Example Design 2 ..... A–3

Additional Information

Revision History Info–1
How to Contact Altera Info–2
Typographic Conventions Info–2

1. About This MegaCore Function

Release Information

Table 1–1 provides information about this release of the Altera® NCO MegaCore® function.

Table 1–1. NCO MegaCore Function Release Information

Item	Description
Version	8.0
Release Date	May 2008
Ordering Code	IP-NCO
Product ID(s)	0014
Vendor ID(s)	6AF7

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore® function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

Device Family Support

MegaCore® functions provide either full or preliminary support for target Altera device families, as described below:

- *Full support* means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs.
- *Preliminary support* means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the NCO MegaCore function to each Altera device family.

Table 1–2. Device Family Support

Device Family	Support Level
Arria™ GX	Full
Cyclone®	Full
Cyclone II	Full
Cyclone III	Full
HardCopy® II	Full
Stratix®	Full
Stratix II	Full
Stratix II GX	Full
Stratix III	Preliminary
Stratix IV	Preliminary
Stratix GX	Full
Other device families	No support

Features

The Altera® NCO MegaCore function implements a numerically controlled oscillator and supports the following features:

- Supports 32-bit precision for angle and magnitude
- The source interface is compatible with the *Avalon® Interface Specification*.
- Easy-to-use IP Toolbench interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Supports multiple NCO architectures:
  - Multiplier-based implementation using DSP blocks or logic elements (LEs), (single cycle and multi-cycle)
  - Parallel or serial CORDIC-based implementation
  - ROM-based implementation using embedded array blocks (EABs), embedded system blocks (ESBs), or external ROM
- Supports single or dual outputs (sine/cosine)
- Allows variable width frequency modulation input
- Allows variable-width phase modulation input
- User-defined frequency resolution, angular precision, and magnitude precision
- Generates simulation files and architecture-specific testbenches
  - VHDL testbench
  - Verilog HDL testbench

  - MATLAB model and testbench
  - Quartus® II Vector Files
- Includes dual-output oscillator and quaternary frequency shift keying (QFSK) modulator example designs

General Description

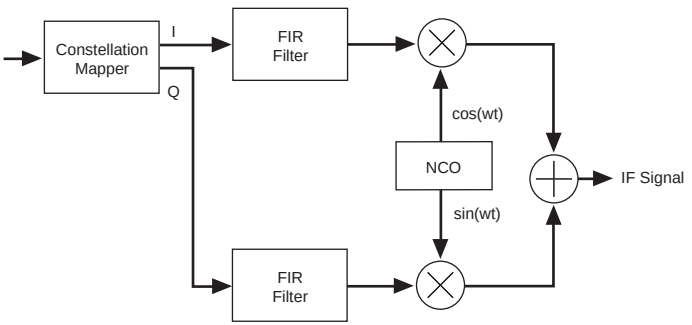
The Altera NCO MegaCore function generates numerically controlled oscillators (NCOs) customized for Altera devices.

You can use the IP Toolbench interface to implement a variety of NCO architectures, including ROM-based, CORDIC-based, and multiplier-based. IP Toolbench also includes time and frequency domain graphs that dynamically display the functionality of the NCO, based on your parameter settings.

A numerically controlled oscillator synthesizes a discrete-time, discrete-valued representation of a sinusoidal waveform. Designers typically use NCOs in communication systems. In such systems, they are used as quadrature carrier generators in I-Q mixers, in which baseband data is modulated onto the orthogonal carriers in one of a variety of ways.

Figure 1–1 shows an NCO used in a simple modulator system.

Figure 1–1. Simple Modulator



Designers also use NCOs in all-digital phase-locked-loops for carrier synchronization in communications receivers, or as standalone frequency shift keying (FSK) or phase shift keying (PSK) modulators. In these applications, the phase or the frequency of the output waveform varies directly according to an input data stream.

MegaCore Verification

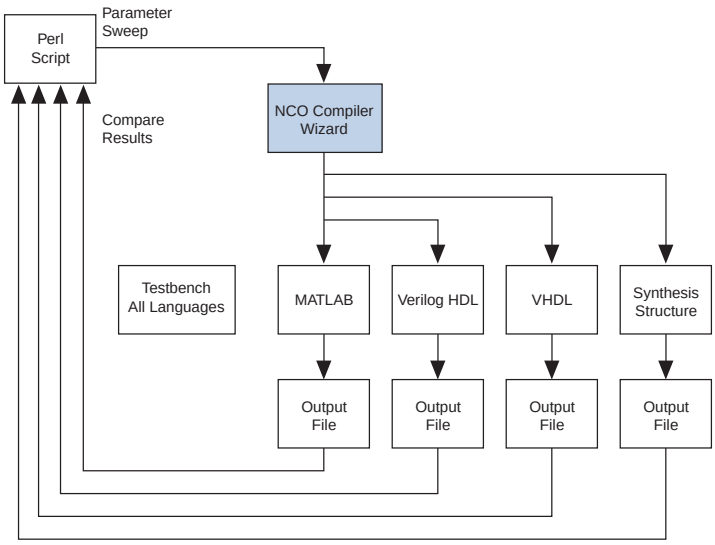
Before releasing a version of the NCO MegaCore function, Altera runs comprehensive regression tests to verify its quality and correctness.

First a custom variation of the NCO MegaCore function is created. Next, Verilog HDL and VHDL IP functional simulation models are exercised by their appropriate testbenches in ModelSim simulators and the results are compared to the output of bit-accurate models in the MATLAB software.

The regression suite covers various parameters such as architecture options, frequency modulation, phase modulation, and precision.

Figure 1–2 shows the regression flow.

Figure 1–2. Regression Flow



Performance and Resource Utilization

This section shows typical expected performance for a NCO MegaCore function using the Quartus II software, version 8.0, and a target f_{MAX} set to 1GHz with Cyclone III and Stratix III devices.

The performance for Stratix IV devices is similar to Stratix III devices. Cyclone III devices use combinational look-up tables (LUTs) and logic registers; Stratix III devices use combinational adaptive look-up tables (ALUTs) and logic registers. It may be possible to significantly reduce memory utilization by setting a lower target f_{MAX}.

Table 1–3 shows performance figures for Cyclone III devices.

Table 1–3. NCO MegaCore Function Performance—Cyclone III Devices

Accumulator Width	Angular Precision	Magnitude Precision	Combinational LUTs	Logic Registers	Memory		9×9 Blocks	f_{MAX} (MHz)
					Bits	M9K		
Large ROM *(1)*								
32	12	12	166	161	458,752	56	—	298
Multiplier-Based *(2)*								
32	16	16	351	287	12,288	2	8	229
Parallel CORDIC *(2)*								
32	14	14	1,173	1,158	—	—	—	300
Small ROM *(2)*								
32	14	16	363	298	61,440	8	—	300

Notes to Table 1–3:
(1) Using EP3C16F484C6 devices.
(2) Using EP3C10F256C6 devices.

Table 1–4 shows performance figures for Stratix III devices.

Table 1–4. NCO MegaCore Function Performance—Stratix III Devices

Accumulator Width	Angular Precision	Magnitude Precision	Combinational ALUTs	Logic Registers	Memory		18×18 Blocks	f_{MAX} (MHz)
					Bits	M9K		
Large ROM *(1)*								
32	12	12	69	149	98,304	12	—	512
Multiplier-Based *(1)*								
32	16	16	117	206	12,288	2	4	498
Parallel CORDIC *(1)*								
32	14	14	1,370	1,536	—	0	—	513
Small ROM *(1)*								
32	14	16	189	298	61,440	8	—	503

Note to Table 1–4:
(1) Using EP3SE50F780C2 devices.

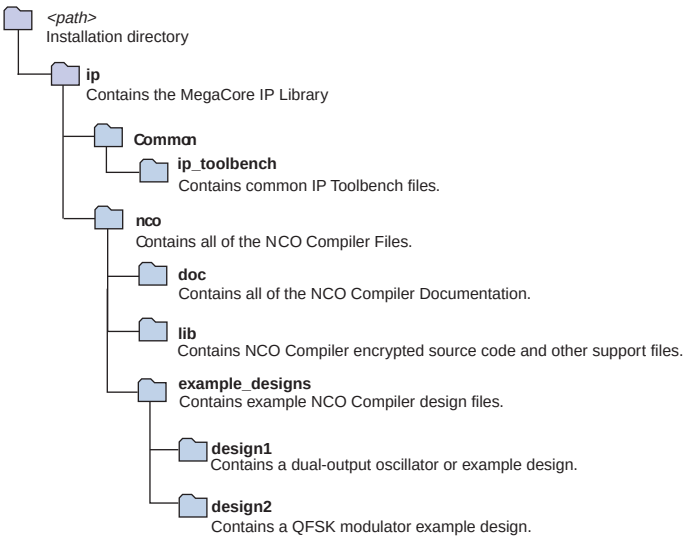
Installation and Licensing

The NCO MegaCore Function is part of the MegaCore® IP Library, which is distributed with the Quartus® II software and downloadable from the Altera® website, www.altera.com.

For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows* or *Quartus II Installation & Licensing for UNIX & Linux Workstations*.

Figure 1–3 shows the directory structure after you install the NCO MegaCore Function, where <*path*> is the installation directory. The default installation directory on Windows is **c:\altera\80**; or on Linux, the default installation directory is **/opt/altera/80**.

Figure 1–3. Directory Structure



OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP℠ megafunction) within your system
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily

- Generate time-limited device programming files for designs that include megafunctions
- Program a device and verify your design in hardware

You only need to purchase a license for the NCO MegaCore function when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license, you can request a license file from the Altera website at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

For more information on OpenCore Plus hardware evaluation, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

OpenCore Plus Time-Out Behavior

OpenCore® Plus hardware evaluation supports the following two operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.

The untethered timeout for the NCO MegaCore function is 1 hour; the tethered timeout value is indefinite.

The `reset_n` signal is forced low when the hardware evaluation time expires. This keeps the NCO MegaCore function permanently in its reset state.

2. Getting Started

Design Flows

The NCO MegaCore® function supports the following design flows:

- **DSP Builder**: Use this flow if you want to create a DSP Builder model that includes a NCO MegaCore function variation.
- **MegaWizard® Plug-In Manager**: Use this flow if you would like to create a NCO MegaCore function variation that you can instantiate manually in your design.

This chapter describes how you can use a NCO MegaCore function in each of these flows. The parameterization is the same in each flow and is described in Chapter 3, Parameter Settings.

After parameterizing and simulating a design in either of these flows, you can compile the completed design in the Quartus II software.

DSP Builder Flow

Altera's DSP Builder product shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

DSP Builder integrates the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB® and Simulink® system-level design tools with Altera Quartus® II software and third-party synthesis and simulation tools. You can combine existing Simulink blocks with Altera DSP Builder blocks and MegaCore function variation blocks to verify system level specifications and perform simulation.

After installing the NCO MegaCore function, a Simulink symbol for the MegaCore function appears in the MegaCore Functions library of the Altera DSP Builder Blockset in the Simulink library browser.

You can use the NCO MegaCore function in the MATLAB/Simulink environment by performing the following steps:

1. Create a new Simulink model.
2. Choose the NCO block from the MegaCore Functions library in the Simulink Library Browser, add it to your model, and give the block a unique name.

3. Double-click on the NCO MegaCore function block in your model to display IP Toolbench and click **Step 1: Parameterize** to parameterize the MegaCore function variation.

For an example of how to set parameters for the NCO MegaCore function, see Chapter 3, Parameter Settings.

4. Click **Step 2: Generate** in IP Toolbench to generate your NCO MegaCore function variation. For information about the generated files, see Table 2–1 on page 2–7.

5. Connect your NCO MegaCore function variation to the other blocks in your model.

6. Simulate the NCO MegaCore function variation in your DSP Builder model.

For more information about the DSP Builder flow, refer to the *Using MegaCore Functions* chapter in the *DSP Builder User Guide*.

When you are using the DSP Builder flow, device selection, simulation, Quartus II compilation and device programming are all controlled within the DSP Builder environment.

DSP Builder supports integration with SOPC Builder using Avalon® Memory-Mapped (Avalon-MM) master/slave and Avalon Streaming (Avalon-ST) source/sink interfaces.

Refer to the *Avalon Interface Specification* for more information about these interface types.

MegaWizard Plug-In Manager Flow

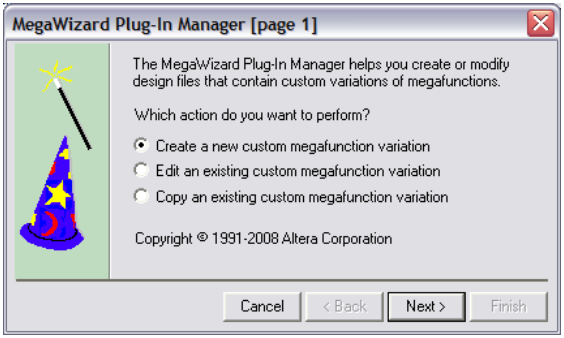
The MegaWizard Plug-in Manager flow allows you to customize a NCO MegaCore function, and manually integrate the MegaCore function variation into a Quartus II design.

Follow the steps below to use the MegaWizard Plug-in Manager flow.

1. Create a new project using the **New Project Wizard** available from the File menu in the Quartus II software.

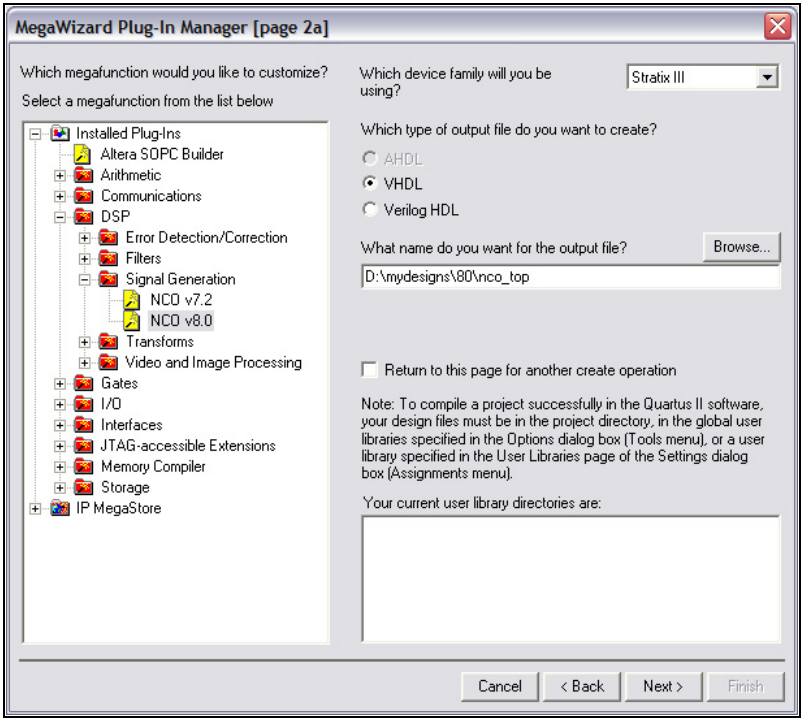
2. Launch **MegaWizard Plug-in Manager** from the Tools menu, and choose the option to create a new custom megafunction variation (Figure 2–1 on page 2–3).

Figure 2–1. MegaWizard Plug-In Manager


MegaWizard Plug-In Manager [page 1]
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.
Which action do you want to perform?
Create a new custom megafunction variation
Edit an existing custom megafunction variation
Copy an existing custom megafunction variation
Copyright © 1991-2008 Altera Corporation
Cancel
< Back
Next >
Finish

3. Click **Next** and choose the `NCO v8.0` from the Signal Generation section in the **Installed Plug-Ins** tab. (Figure 2–2).

Figure 2–2. Selecting the MegaCore Function


MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
Installed Plug-Ins
Altera SOPC Builder
Arithmetic
Communications
DSP
Error Detection/Correction
Filters
Signal Generation
NCO v7.2
NCO v8.0
Transforms
Video and Image Processing
Gates
I/O
Interfaces
JTAG-accessible Extensions
Memory Compiler
Storage
IP MegaStore
Which device family will you be using?
Stratix III
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
Browse...
D:\mydesigns\80\nco_top
Return to this page for another create operation
Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish

4. Check that the device family is the same as you specified in the **New Project Wizard**.

Refer to the Quartus II help for more information about the MegaWizard Plug-In Manager.

5. Choose the top level output file name for your MegaCore function variation and click **Next** to launch IP Toolbench (see Figure 2–3).

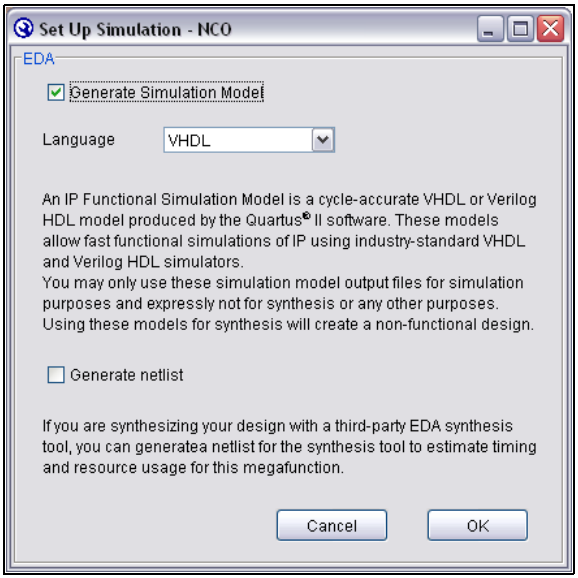
Figure 2–3. IP Toolbench—Parameterize

6. Click **Step 1: Parameterize** in IP Toolbench to display the **Parameterize - NCO** window. Use this dialog box to specify the required parameters for the MegaCore function variation.

For an example of how to set parameters for the NCO MegaCore function, see Chapter 3, Parameter Settings.

7. Click **Step 2: Setup Simulation** in IP Toolbench to display the **Set Up Simulation - NCO** page (Figure 2–4 on page 2–5).

Figure 2–4. Set Up Simulation


Set Up Simulation - NCO
EDA
Generate Simulation Model
Language
VHDL
An IP Functional Simulation Model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus® II software. These models allow fast functional simulations of IP using industry-standard VHDL and Verilog HDL simulators.
You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis will create a non-functional design.
Generate netlist
If you are synthesizing your design with a third-party EDA synthesis tool, you can generatea netlist for the synthesis tool to estimate timing and resource usage for this megafunction.
Cancel
OK

8. Turn on **Generate Simulation Model** to create an IP functional model.

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.

CAUTION

Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

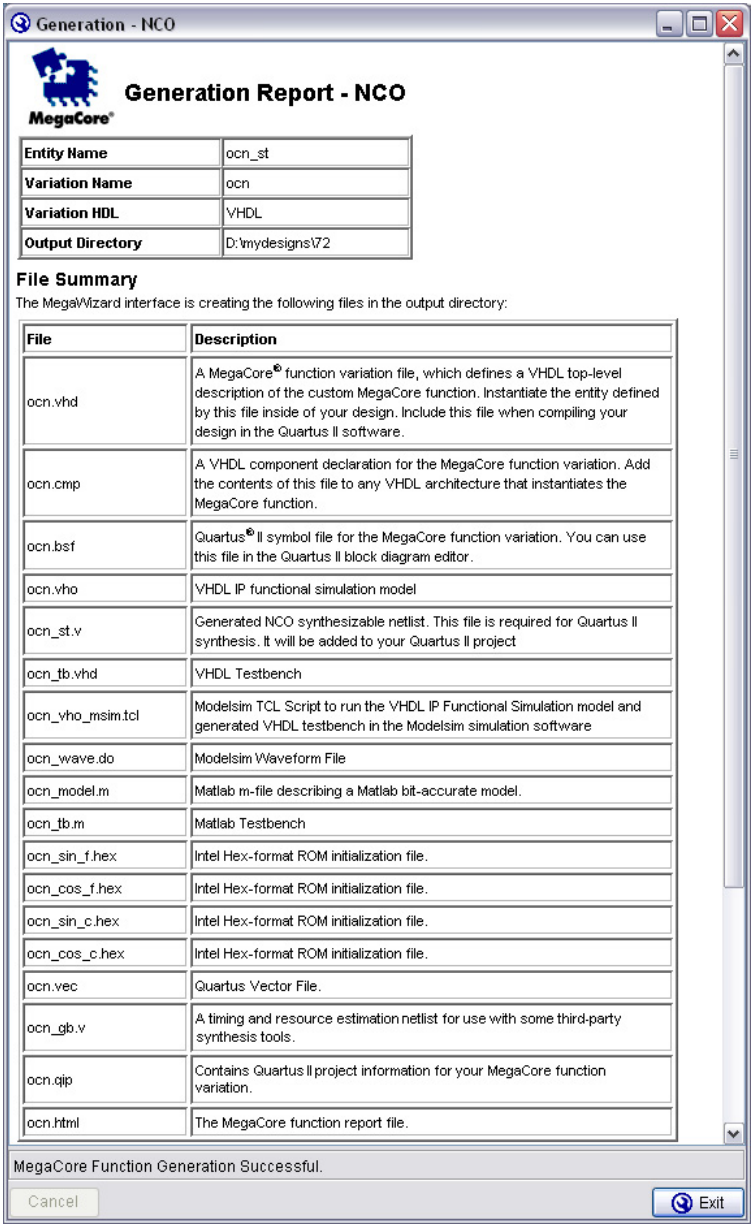
9. Choose the required language from the **Language** list.

10. Some third-party synthesis tools can use a netlist that contains only the structure of the MegaCore function, but not detailed logic, to optimize performance of the design that contains the MegaCore function. If your synthesis tool supports this feature, turn on **Generate netlist**.

11. Click **Step 3: Generate** in IP Toolbench to generate your MegaCore function variation and supporting files. The generation phase may take several minutes to complete. The generation progress and status is displayed in a report window.

Figure 2–5 shows the generation report.

Figure 2–5. Generation Report - NCO MegaCore function


Generation - NCO
MegaCore
Generation Report - NCO
Entity Name
ocn_st
Variation Name
ocn
Variation HDL
VHDL
Output Directory
D:\mydesigns\72
File Summary
The MegaWizard interface is creating the following files in the output directory:
File
Description
ocn.vhd
A MegaCore® function variation file, which defines a VHDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
ocn.cmp
A VHDL component declaration for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
ocn.bsf
Quartus® II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
ocn.vho
VHDL IP functional simulation model
ocn_st.v
Generated NCO synthesizable netlist. This file is required for Quartus II synthesis. It will be added to your Quartus II project
ocn_tb.vhd
VHDL Testbench
ocn_vho_msim.tcl
Modelsim TCL Script to run the VHDL IP Functional Simulation model and generated VHDL testbench in the Modelsim simulation software
ocn_wave.do
Modelsim Waveform File
ocn_model.m
Matlab m-file describing a Matlab bit-accurate model.
ocn_tb.m
Matlab Testbench
ocn_sin_f.hex
Intel Hex-format ROM initialization file.
ocn_cos_f.hex
Intel Hex-format ROM initialization file.
ocn_sin_c.hex
Intel Hex-format ROM initialization file.
ocn_cos_c.hex
Intel Hex-format ROM initialization file.
ocn.vec
Quartus Vector File.
ocn_qb.v
A timing and resource estimation netlist for use with some third-party synthesis tools.
ocn.qip
Contains Quartus II project information for your MegaCore function variation.
ocn.html
The MegaCore function report file.
MegaCore Function Generation Successful.
Cancel
Exit

Generated Files

Table 2–1 describes the generated files and other files that may be in your project directory. The names and types of files specified in the report vary based on whether you created your design with VHDL or Verilog HDL.

Table 2–1. IP Toolbench Files (Part 1 of 2)

Filename *Note (1)*, *Note (2)*	Description
<*entity name*>**.v**	Generated synthesizable netlist. This file is required for Quartus II synthesis. It will be added to your Quartus II project.
<*variation name*>**_vho_msim.tcl** <*variation name*>**_vo_msim.tcl**	ModelSim TCL Script that runs the VHDL or Verilog HDL IP functional simulation model and generated VHDL or Verilog testbench in the ModelSim simulation software.
<*variation name*>**_tb.v or** <*variation name*>**_tb.vhd**	A VHDL or Verilog HDL testbench file for the MegaCore function variation. The VHDL file is generated when a VHDL top level has been chosen or the Verilog HDL file when a Verilog HDL top level has been chosen.
<*variation name*>**.bsf**	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<*variation name*>**.cmp**	A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
<*variation name*>**.html**	MegaCore function report file.
<*variation name*>**.qip**	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
<*variation name*>**.vec**	Quartus II Vector File.
<*variation name*>**.vhd** or **.v**	A MegaCore function variation file, which defines a Verilog HDL or VHDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside your design. Include this file when compiling your design in the Quartus II software.
<*variation name*>**.vho** or <*variation name*>**.vo**	VHDL or Verilog HDL IP functional simulation model.
<*variation name*>**_bb.v**	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<*variation name*>**_cos_c.hex**, <*variation name*>**_cos_f.hex**, <*variation name*>**_sin_c.hex**, <*variation name*>**_sin_f.hex**	Intel Hex-format ROM initialization files.
<*variation name*>**_gb.v**	A timing and resource estimation netlist for use in some third-party synthesis tools.
<*variation name*>**_model.m**	MATLAB m-file describing a MATLAB bit-accurate model.

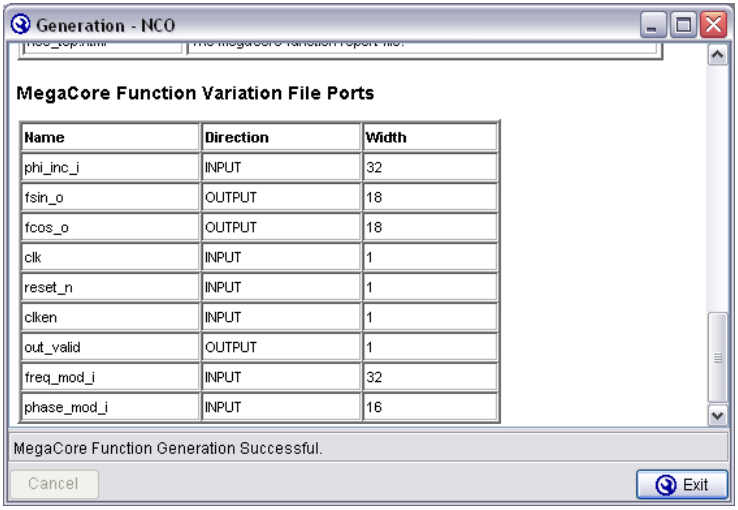
Table 2–1. IP Toolbench Files (Part 2 of 2)	
Filename *Note (1), Note (2)*	**Description**
<*variation name*>_**nativelink.tcl**	A Tcl script that can be used to assign NativeLink simulation testbench settings to the Quartus II project.
<*variation name*>_**tb.m**	MATLAB testbench file.
<*variation name*>_**wave.do**	ModelSim Waveform file.

Notes to Table 2–1:
(1) <*variation name*> is a prefix variation name supplied automatically by IP Toolbench.
(2) The <*entity name*> prefix is added automatically. The VHDL code for each MegaCore instance is generated dynamically when you click **Finish** so that the <entity name> is different for every instance. It is generated from the <*variation name*> by appending **_st.**

The generation report also lists the MegaCore function variation signals (see Figure 2–6).

For a full description of the signals supported on external ports for your MegaCore function variation, refer to Table 4–4 on page 4–13.

Figure 2–6. Port Lists in the Generation Report



12. After you review the generation report, click **Exit** to close IP Toolbench. Then click **Yes** on the **Quartus II IP Files** prompt to add the .**qip** file describing your custom MegaCore function variation to the current Quartus II project.

Simulate the Design

To simulate your design, use the IP functional simulation models generated by IP Toolbench. The IP functional simulation model is either a **.vo** or **.vho** file, depending on the output language you specified. Compile the **.vo** or **.vho** file in your simulation environment to perform functional simulation of your custom variation of the MegaCore function.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

Simulating in Third-Party Simulation Tools Using NativeLink

You can perform a simulation in a third-party simulation tool from within the Quartus II software, using NativeLink.

The Tcl script file **<***variation name***>_nativelink.tcl** can be used to assign default NativeLink testbench settings to the Quartus II project. To perform a simulation in the Quartus II software using NativeLink, perform the following steps:

1. Create a custom MegaCore function variation as described earlier in this chapter but ensure you specify your variation name to match the Quartus II project name.
2. Check that the absolute path to your third-party EDA tool is set in the **Options** page under the Tools menu in the Quartus II software.
3. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu, click **Tcl scripts**. Select the Tcl script <*variation name*>**_nativelink.tcl** and click **Run**. Check for a message confirming that the Tcl script was successfully loaded.
5. On the Assignments menu, click **Settings**, expand **EDA Tool Settings**, and select **Simulation**. Select a simulator under **Tool name** then in **NativeLink Settings**, select **Compile test bench** and click **Test Benches**. Confirm that appropriate testbench settings have been applied to the Quartus II project.
6. On the Tools menu, point to **EDA Simulation Tool** and click **Run EDA RTL Simulation**.

Refer to *Simulating Altera IP Using the Quartus II NativeLink Feature* in volume 3 of the *Quartus II Handbook* for more information.

Simulating the Design in ModelSim

To simulate your design with the MegaWizard-generated ModelSim Tcl script, change your ModelSim working directory to the project directory specified in "Selecting the MegaCore Function" on page 2–3, and run the MegaWizard-generated Tcl script.

- If you selected VHDL as your functional simulation language, run the Tcl script <*variation_name*>**_vho_msim.tcl**.
- If you selected Verilog HDL as your functional simulation language, run the Tcl script <*variation_name*>**_vo_msim.tcl**

The Tcl script creates a ModelSim project, maps the libraries, compiles the top-level design and associated testbench, and then outputs the simulation results to the waveform viewer.

Compile the Design and Program a Device

You can use the Quartus II software to compile your design.

To compile your design, follow these steps:

1. If you are using the Quartus II software to synthesize your design, skip to **Step 2**. If you are using a third-party synthesis tool to synthesize your design, follow these steps:
   a. Set a black-box attribute for your MegaCore function custom variation before you synthesize the design. Refer to Quartus II Help for instructions on setting black-box attributes for synthesis tools.
   b. Run the synthesis tool to produce an EDIF netlist file (**.edf**) or a Verilog Quartus Mapping (VQM) file (**.vqm**) for input to the Quartus II software.
   c. Add the EDIF or VQM file to your Quartus II project.
2. Choose **Start Compilation** (Processing menu) in Quartus II software.

After a successful compilation, you can program the targeted Altera device and verify the design in hardware.

Refer to Quartus II Help for instructions on compiling and programming your design.

3. Parameter Settings

This chapter gives an example of how to parameterize a NCO MegaCore® function and describes the available parameters.

The **Parameterize - NCO** pages provide the same options whether they have been opened from the DSP Builder or MegaWizard Plug-In Manager flow.

For information about opening the parameterization pages, refer to the "Design Flows" section in Chapter , Design Flows.

The user interface only allows you to select legal combinations of parameters, and warns you of any invalid configurations.

Setting Parameters

To parameterize your NCO MegaCore function, follow these steps:

1. With the **Parameters** tab selected, specify the generation algorithm, precisions, phase dithering, and generated output frequency parameters.

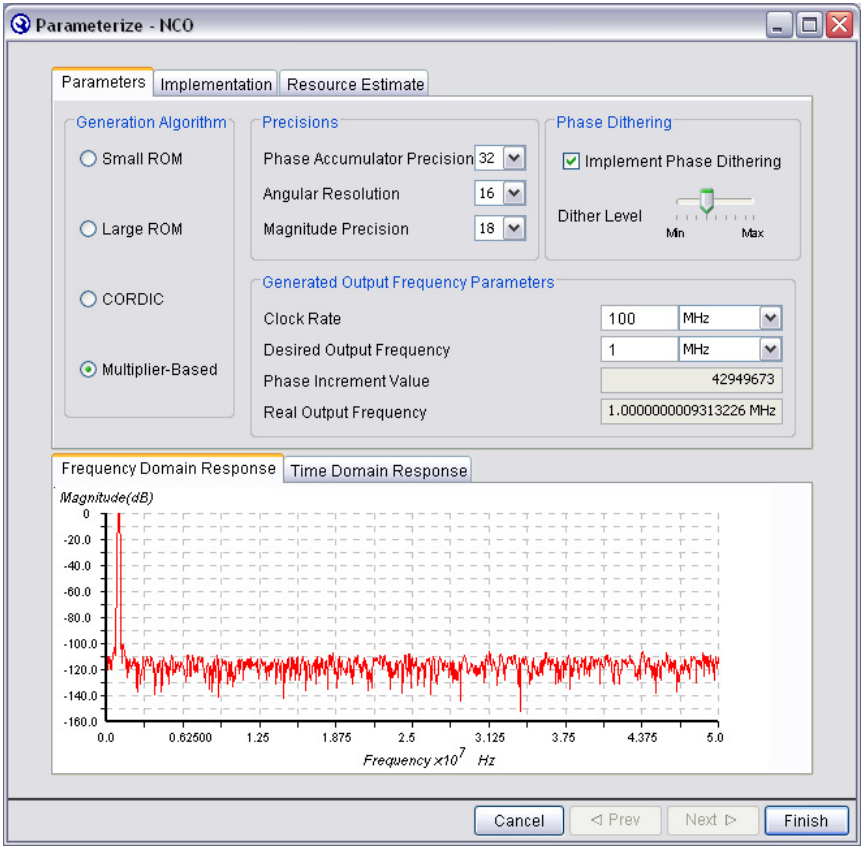
   As you adjust these parameters, you can graphically view the effects on the NCO MegaCore function in the **Frequency Domain Response** and **Time Domain Response** tabs. See Figure 3–1 on page 3–2.

   The NCO MegaCore function generates the spectral plot shown in Figure 3–1 by computing a 2,048-point fast Fourier transform (FFT) of bit-accurate time-domain data. Before performing the FFT, IP Toolbench applies a Kaiser window of length 2,048 to the data.

   You can zoom into the view by pressing the left mouse key on the plot drawing a box around the area of interest. Right-click the plot to restore the view to its full range.

Refer to "Architectures" on page 4–5 and "Phase Dithering" on page 4–9 for more information about these parameter options.

Figure 3–1. Parameterize Tab



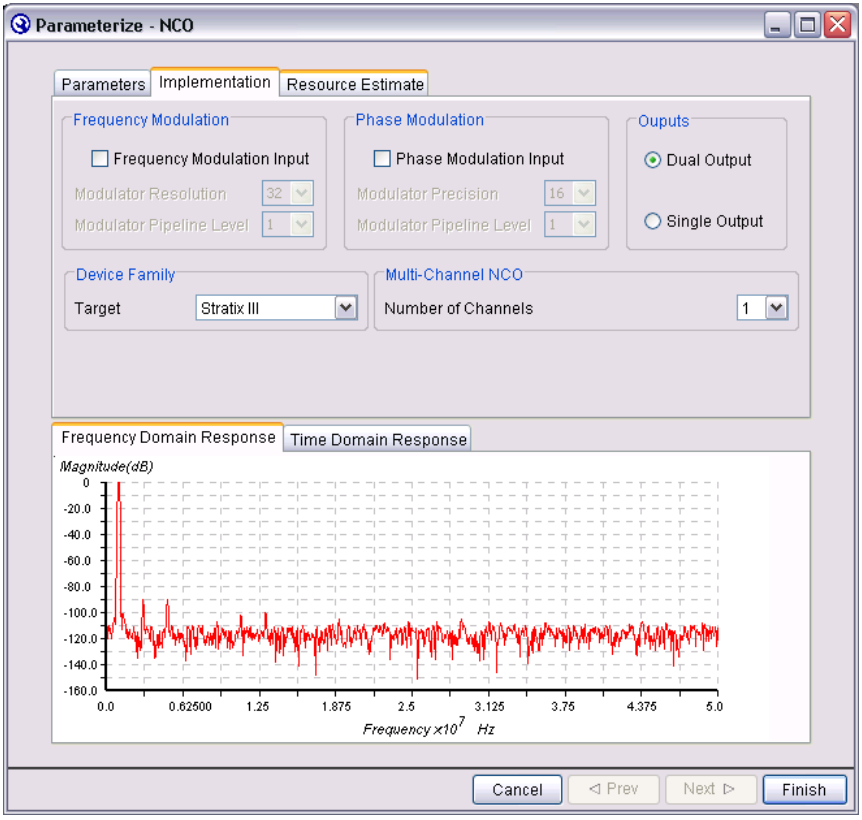
2. Click the **Implementation** tab when you are finished setting the general parameters.

3. With the **Implementation** tab selected, specify the frequency modulation, phase modulation, and outputs; choose the target device family.

   For some algorithms (for example, multiplier-based), you can also make device-specific settings such as whether to implement the NCO MegaCore function in logic elements (LEs) or other hardware. The **Implementation** tab displays the corresponding options available for the selected algorithm in the **Parameters** tab.

Figure 3–2 shows the implementation parameter options when you specify the **Small ROM** or **Large ROM** algorithm.

Figure 3–2. Implementation Tab - Large ROM Algorithm



Refer to "Frequency Modulation" on page 4–8 and "Phase Modulation" on page 4–9 for more information about these parameter options.

Figure 3–3 shows implementation parameter options when the **CORDIC** algorithm is specified.

With the CORDIC algorithm, you can choose a parallel or serial CORDIC implementation.

Figure 3–3. Implementation Tab - CORDIC Algorithm

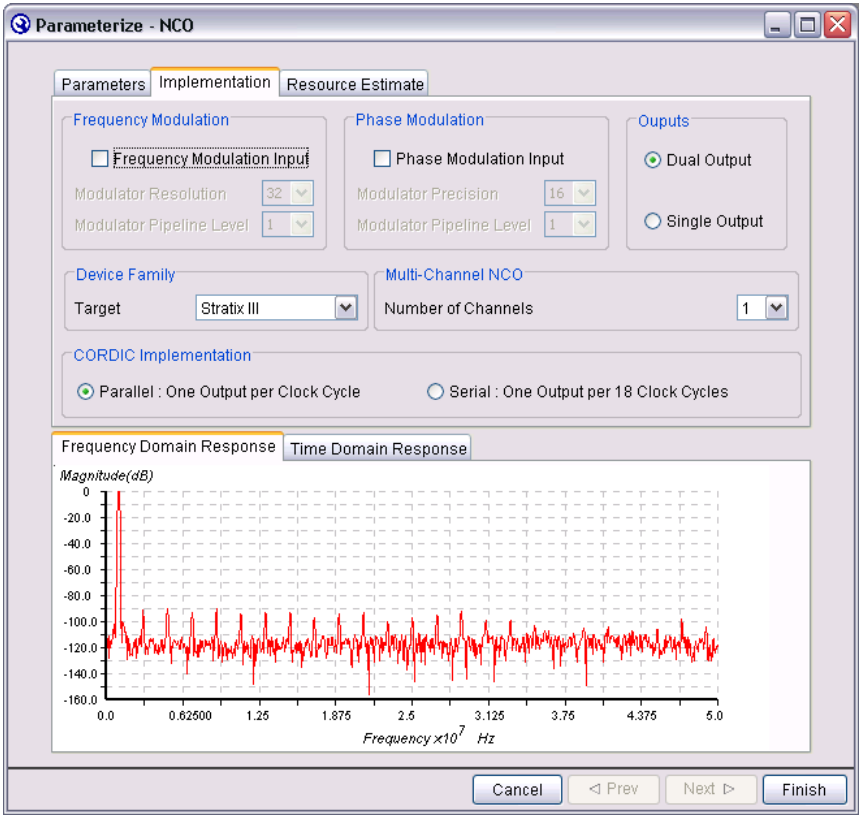


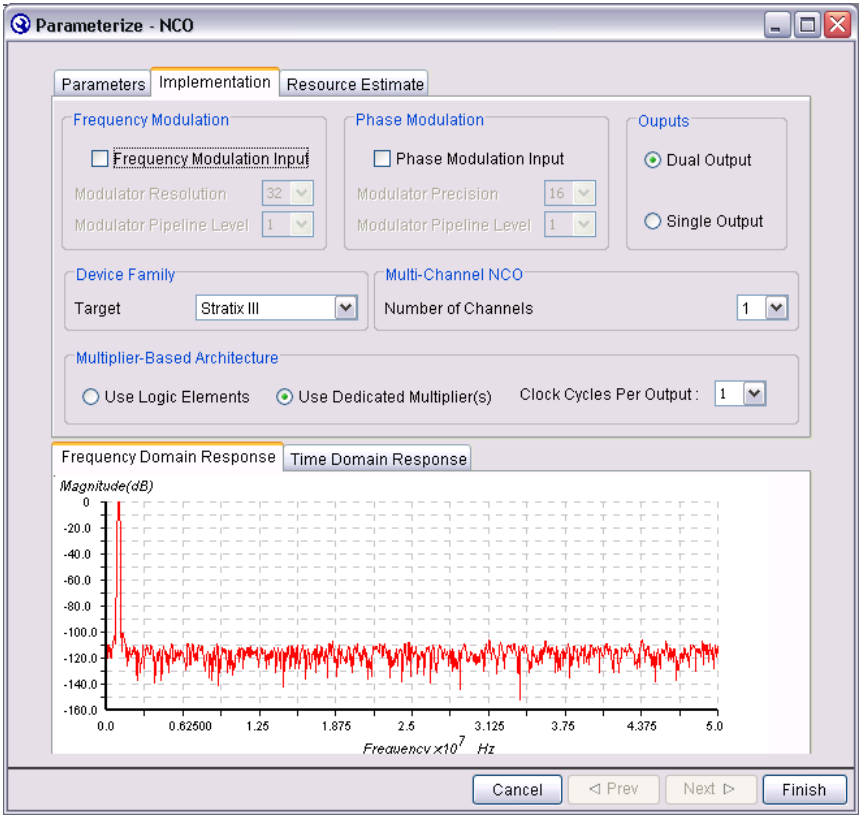
Figure 3–4 on page 3–5 shows the implementation parameter options when you specify the **Multiplier-Based** algorithm.

If you target the Stratix IV, Stratix III, Stratix II, Stratix GX, or Stratix device families, you can choose whether to implement the multiplier-based algorithm using logic elements or dedicated multipliers. If you specify multiplier-based and do not target Stratix device families, the Quartus II software implements the NCO MegaCore function using logic elements.

CAUTION

Do not change the **Target** device family in the **Implementation** page. The device family is automatically set to the value that was specified in the Quartus II or DSP Builder software and the generated HDL for your MegaCore function variation may be incorrect if this value is changed in IP Toolbench.

Figure 3–4. Implementation Tab - Multiplier-Based Algorithm

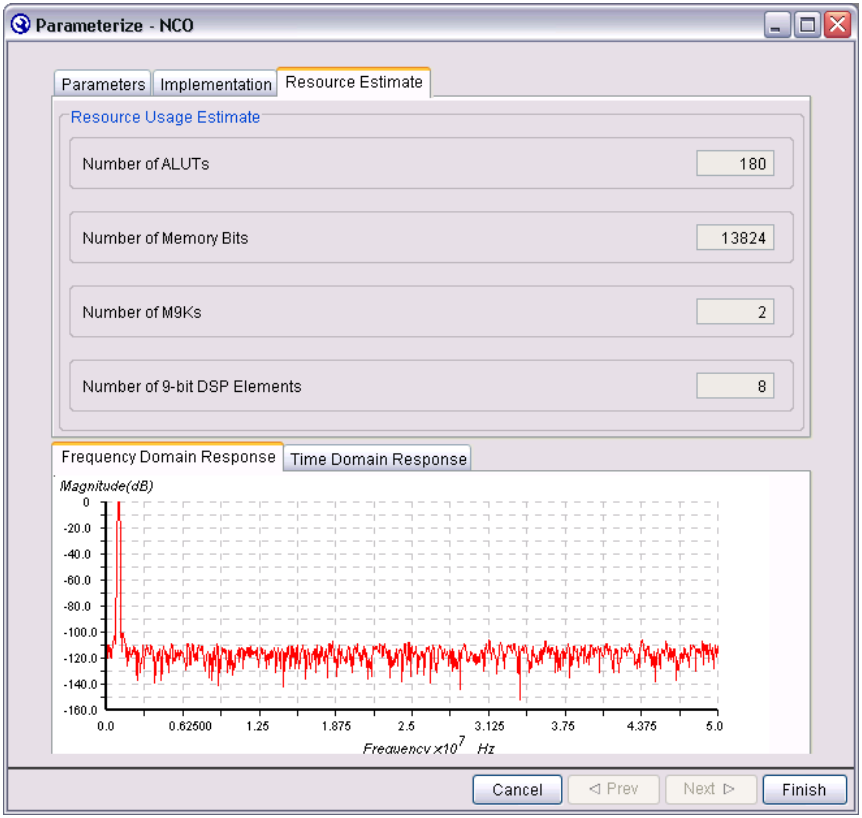


4. Click the **Resource Estimate** tab when you are finished setting the implementation parameter options.

   The NCO MegaCore function dynamically estimates the resource usage of your custom NCO MegaCore function variation based on the parameters specified. See Figure 3–5.

   Arria GX, Stratix II, Stratix II GX, Stratix III and Stratix IV devices use adaptive look-up tables (ALUTs); other devices use logic elements (LEs).

Figure 3–5. Resource Estimate Tab



5. Click **Finish** when you are finished viewing the resource estimates.

Parameter Descriptions

This section describes the NCO MegaCore function parameters, which can be set in the user interface (see "Setting Parameters" on page 3–1).

Table 3–1 shows the parameters that can be set in the **Parameters** page.

Table 3–1. NCO MegaCore Function Parameters Page (Part 1 of 2)

Parameter	Value	Description
Generation Algorithm	Small ROM, Large ROM, CORDIC, **Multiplier-Based**	Choose the required algorithm.

Table 3–1. NCO MegaCore Function Parameters Page (Part 2 of 2)

Parameter	Value	Description
Phase Accumulator Precision	4–64 Default = **32**	Choose the required phase accumulator precision. *Note (1)*
Angular Resolution	4–24 or 32 Default = **16**	Choose the required angular resolution. *Note (2)*,
Magnitude Precision	10–32 Default = **18**	Choose the required magnitude precision.
Implement Phase Dithering	**On** or Off	Turn on to implement phase dithering.
Dither Level	Min–Max	When phase dithering is enabled you can use the slider control to adjust the dither level between its minimum and maximum values,
Clock Rate	1–999 MHz, kHz, Hz, mHz Default = **100 MHz**	You can choose the clock rate using units of MegaHertz, kiloHertz, Hertz or milliHertz.
Desired Output Frequency	1–999 MHz, kHz, Hz, mHz Default = **1 MHz**	You can choose the desired output frequency using units of MegaHertz, kiloHertz, Hertz or milliHertz.
Phase Increment Value	—	Displays the phase increment value calculated from the clock rate and desired output frequency.
Real Output Frequency	—	Displays the calculated value of the real output frequency.

Note for Table 3–1:
(1) The phase accumulator precision must be greater than or equal to the specified angular resolution.
(2) The maximum value is 24 for small and large ROM algorithms; 32 for CORDIC and multiplier-based algorithms.

Table 3–2 shows the parameters that can be set in the **Implementation** page.

Table 3–2. NCO MegaCore Function Implementation Page (Part 1 of 2)

Parameter	Value	Description
Frequency Modulation input	On or **Off**	You can optionally enable the frequency modulation input.
Modulator Resolution	4–64 Default = **32**	Choose the modulator resolution for the frequency modulation input.
Modulator Pipeline Level	1, 2 Default = **1**	Choose the modulator pipeline level for the frequency modulation input.
Phase Modulation Input	On or **Off**	You can optionally enable the phase modulation input.
Modulator Precision	4–32 Default = **16**	Choose the modulator precision for the phase modulation input.

Table 3–2. NCO MegaCore Function Implementation Page (Part 2 of 2)

Parameter	Value	Description
Modulator Pipeline Level	1, 2 Default = **1**	Choose the modulator pipeline level for the phase modulation input.
Outputs	**Dual Output**, Single Output	Choose whether to use a dual or single output.
Device Family Target	Stratix IV, Stratix III, Stratix II, Stratix II GX, Arria GX, Stratix, Stratix GX, Cyclone III, Cyclone II, Cyclone	Displays the target device family. The target device family is pre-selected by the value specified in the Quartus II or DSP Builder software. The HDL that is generated for your MegaCore function variation may be incorrect if you change the device family target in IP Toolbench.
Number of Channels	1–8 Default = **1**	Choose the number of channels when you want to implement a multi-channel NCO.
CORDIC Implementation	Parallel, Serial	When the CORDIC generation algorithm is selected on the Parameters page, you can choose to use a parallel (one output per clock cycle) or serial (one output per 18 clock cycles) implementation.
Multiplier-Based Architecture	Logic Elements, Dedicated Multipliers	When the multiplier-based algorithm is selected on the Parameters page, you can choose whether to use logic elements or dedicated multipliers and choose the number of clock cycles per output.
Clock Cycles Per Output	1, 2 Default = **1**	When the multiplier-based algorithm is selected on the Parameters page, you can choose 1 or 2 clock cycles per output.

Table 3–3 shows the parameters that are displayed in the **Resource Estimate** page.

Table 3–3. NCO MegaCore Function Resource Estimate Page

Parameter	Description
Number of ALUTs/LEs	Displays the number of adaptive look-up tables or logic elements. *Note (1)*
Number of Memory Bits	Displays the number of memory bits.
Number of M9Ks/M4Ks	Displays the number of M9K or M4K RAM blocks. *Note (2)*
Number of 9-bit DSP Elements	Displays the number of 9-bit DSP elements.

Note for Table 3–3:
(1) Stratix GX, Stratix, Cyclone III, Cyclone II and Cyclone devices use LEs; all other devices use ALUTs.
(2) Stratix IV, Stratix III, and Cyclone III devices use M9K RAM blocks; all other devices use M4K blocks.

4. Functional Description

Numerically Controlled Oscillators

A numerically controlled oscillator (NCO) synthesizes a discrete-time, discrete-valued representation of a sinusoidal waveform. There are many ways to synthesize a digital sinusoid. For example, a popular method is to accumulate phase increments to generate an angular position on the unit circle and then use the accumulated phase value to address a ROM look-up table that performs the polar-to-cartesian transformation. You can reduce the ROM size by using multipliers. Multipliers provide an exponential decrease in memory usage for a given precision but require more logic.

Another method uses the coordinate rotation digital computer (CORDIC) algorithm to determine, given a phase rotation, the sine and cosine values iteratively. The CORDIC algorithm takes an accumulated phase value as input and then determines the cartesian coordinates of that angle by a series of binary shifts and compares.

For more information about the CORDIC algorithm, refer to *A Survey of CORDIC Algorithms for FPGAs* (see Reference 1 on page 4–14).

In all methods, the frequency at which the phase increment accumulates and the size of that input phase increment relative to the maximum size of the accumulator directly determines the normalized sinusoidal frequency (see Equation 2 on page 4–3).

When deciding which NCO implementation to use in programmable logic, you should consider several parameters, including the spectral purity, frequency resolution, performance, throughput, and required device resources. Often, you need to consider the trade-offs between some or all of these parameters.

Spectral Purity

Typically, the spectral purity of an oscillator is measured by its signal-to-noise ratio (SNR) and its spurious free dynamic range (SFDR). The SNR of a digitally synthesized sinusoid is a ratio of the signal power relative to the unavoidable quantization noise inherent in its discrete-valued representation. SNR is a direct result of the finite precision with which NCO represents the output sine and cosine waveforms. Increasing the output precision results in an increased SNR.

The following equation estimates the SNR of a given sinusoid with output precision *b*:

$$SNR = 6b - 1.8 \text{ (dB)} \quad (1)$$

Each additional bit of output precision leads to an additional 6 dB in SNR.

The SFDR of a digital sinusoid is the power of the primary or desired spectral component relative to the power of its highest-level harmonic component in the spectrum. Harmonic components manifest themselves as spikes or spurs in the spectral representation of a digital sinusoid and occur at regular intervals and are also a direct consequence of finite precision. However, the effect of the spurs is often severe because they can cause substantial inter-modulation products and undesirable replicas of the mixed signal in the spectrum, leading to poor reconstruction of the signal at the receiver.

The direct effect of finite precision varies between architectures, but the effect is augmented because, due to resource usage constraints, the NCO does not usually use the full accumulator precision in the polar-to-cartesian transformation. You can mitigate truncation effects with phase dithering, in which the truncated phase value is randomized by a sequence. This process removes some of the periodicity in the phase, reducing the spur magnitude in the sinusoidal spectrum by up to 12 dB.

The NCO MegaCore function's graphical spectral analysis allows you to view the effects as you change parameters without regenerating the IP Toolbench output files and re-running simulation.

Refer to "Setting Parameters" on page 3–1 for information about how you can view the effects of changing the generation algorithm, precision, phase dithering and generated output frequency parameters.

Maximum Output Frequency

The maximum frequency sinusoid that an NCO can generate is bounded by the Nyquist criterion to be half the operating clock frequency. Additionally, the throughput affects the maximum output frequency of the NCO. If the NCO outputs a new set of sinusoidal values every clock cycle, the maximum frequency is the Nyquist frequency. If, however, the implementation requires additional clock cycles to compute the values, the maximum frequency must be further divided by the number of cycles per output.

Functional Description

The NCO MegaCore function allows you to generate a variety of NCO architectures. You can create your custom NCO using an IP Toolbench-driven interface that includes both time- and frequency-domain analysis tools. The custom NCO outputs a sinusoidal waveform in two's complement representation. The waveform for the generated sine wave is defined by the following equation:

$$s(nT) = A\sin\left[2\pi((f_O + f_{FM})nT + \phi_{PM} + \phi_{DITH})\right] \quad (2)$$

where:

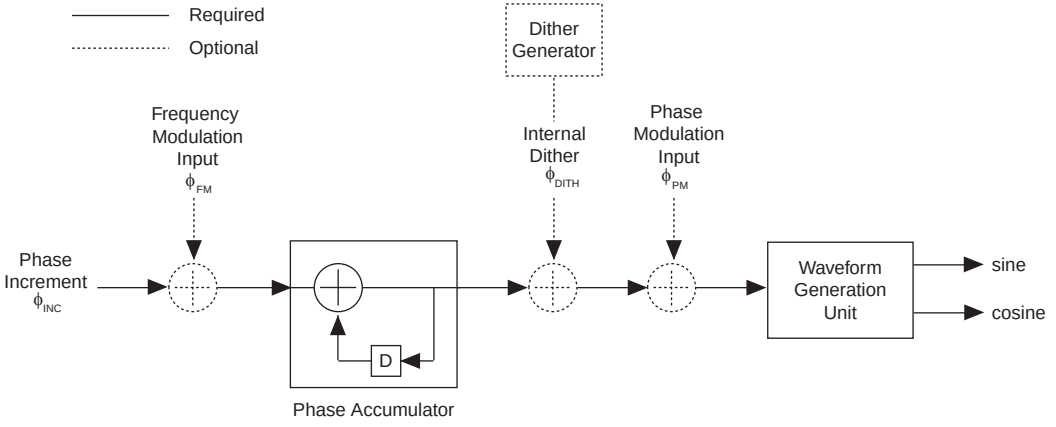
- T is the operating clock period
- f_O is the unmodulated output frequency based on the input value ϕ_{INC}
- f_{FM} is a frequency modulating parameter based on the input value ϕ_{FM}
- ϕ_{PM} is derived from the phase modulation input value P and the number of bits (P_{width}) used for this value by the equation:

$$\phi_{PM} = \frac{P}{2^{P_{width}}}$$

- ϕ_{DITH} is the internal dithering value
- A is 2^{N-1} where N is the magnitude precision (and N is an integer in the range 10–32)

Figure 4–1 shows a block diagram of a generic NCO.

Figure 4–1. NCO Block Diagram



The generated output frequency, f_o for a given phase increment, ϕ_{inc} is determined by the equation:

(3)
$$f_o = \frac{\phi_{inc} \ f_{clk}}{2^M} \text{ Hz}$$

where M is the *accumulator precision* and f_{clk} is the clock frequency.

The minimum possible output frequency waveform is generated for the case where ϕ_{inc}= 1. This case is also the smallest observable frequency at the output of the NCO, also known as the *frequency resolution* of the NCO, f_{res} given in Hz by the equation:

(4)
$$f_{res} = \frac{f_{clk}}{2^M} \text{ Hz}$$

For example, if a 100 MHz clock drives an NCO with an accumulator precision of 32 bits, the frequency resolution of the oscillator is 0.0233 Hz. If you want an output frequency of 6.25 MHz from this oscillator, then you should apply an input phase increment of:

(5)
$$\frac{6.25 \times 10^6}{100 \times 10^6} \times 2^{32} = 268435456$$

The NCO MegaCore function automatically calculates this value, given the parameters you choose. IP Toolbench also sets the value of the phase increment in all testbenches and vector source files it generates.

Similarly, the generated output frequency, f_{FM} for a given frequency modulation increment, ϕ_{FM} is determined by the equation:

(6)
$$f_{FM} = \frac{\phi_{FM} \ f_{clk}}{2^F} \text{ Hz}$$

where F is the *modulator resolution*.

The *angular precision* of an NCO is the phase angle precision before the polar-to-cartesian transformation. The *magnitude precision* is the precision to which the sine and/or cosine of that phase angle can be represented. The effects of reduction or augmentation of the angular, magnitude, accumulator precision on the synthesized waveform vary across NCO architectures and for different f_o/f_{clk} ratios.

You can view these effects in the NCO time and frequency domain graphs as you change the NCO MegaCore function parameters.

Architectures

The NCO MegaCore function supports large ROM, small ROM, CORDIC, and multiplier-based architectures.

Large ROM Architecture

Use the large ROM architecture if your design requires very high speed sinusoidal waveforms, and your design can use large quantities of internal memory. In this architecture, the ROM stores the full 360 degrees of both the sine and cosine waveforms. The output of the phase accumulator addresses the ROM.

Because the internal memory holds all possible output values for a given angular and magnitude precision, the generated waveform has the highest spectral purity for that parameter set (assuming no dithering). The large ROM architecture also uses the fewest logic elements (LEs) for a given set of precision parameters.

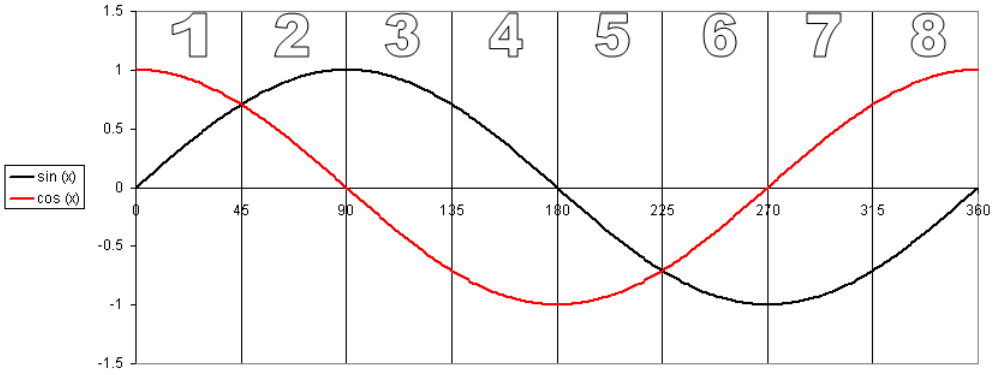
Small ROM Architecture

If low LE usage and high output frequency are a high priority for your system, use the small ROM architecture to reduce your internal memory usage. In a small ROM architecture, the device memory only stores 45 degrees of the sine and cosine waveforms. All other output values are derived from these values based on the position of the rotating phasor on the unit circle as shown in Table 4–1 and Table 4–2.

Table 4–1. Derivation of Output Values

Position in Unit Circle	Range for Phase x	sin(x)	cos(x)
1	$0 <= x < \pi/4$	sin(x)	cos(x)
2	$\pi/4 <= x < \pi/2$	$\cos(\pi/4x)$	$\sin(\pi/2-x)$
3	$\pi/2 <= x < 3\pi/4$	$\cos(x-\pi/2)$	$-\sin(x-\pi/2)$
4	$3\pi/4 <= x < \pi$	$\sin(\pi-x)$	$-\cos(\pi-x)$
5	$\pi <= x < 5\pi/4$	$-\sin(x-\pi)$	$-\cos(x-\pi)$
6	$5\pi/4 <= x < 3\pi/2$	$-\cos(3\pi/2-x)$	$-\sin(3\pi/2-x)$
7	$3\pi/2 <= x < 7\pi/4$	$-\cos(x-3\pi/2)$	$\sin(x-3\pi/2)$
8	$7\pi/4 <= x < 2\pi$	$-\sin(2\pi-x)$	$\cos(2\pi-x)$

Figure 4–2. Derivation of Output Values



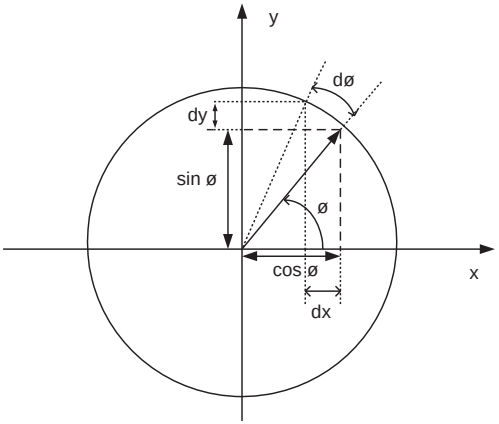
Because a small ROM implementation is more likely to have periodic value repetition, the resulting waveform's SFDR is generally lower than that of the large ROM architecture. However, you can often mitigate this reduction in SFDR by using phase dithering. See ""Phase Dithering" on page 4–9 for more information on this option.

CORDIC Architecture

The CORDIC algorithm, which can calculate trigonometric functions such as sine and cosine, provides a high-performance solution for very-high precision oscillators in systems where internal memory is at a premium. The CORDIC algorithm is based on the concept of complex phasor rotation by multiplication of the phase angle by successively smaller constants. In digital hardware, the multiplication is by powers of two only. Therefore, the algorithm can be implemented efficiently by a series of simple binary shift and additions/subtractions.

In an NCO, the CORDIC algorithm computes the sine and cosine of an input phase value by iteratively shifting the phase angle to approximate the cartesian coordinate values for the input angle. At the end of the CORDIC iteration, the *x* and *y* coordinates for a given angle represent the cosine and sine of that angle, respectively. See Figure 4–3 on page 4–7.

Figure 4–3. CORDIC Rotation for Sine & Cosine Calculation



With the NCO MegaCore function, you can choose between parallel (unrolled) and serial (iterative) CORDIC architectures:

- You an use the parallel CORDIC architecture to create a very high-performance, high-precision oscillator—implemented entirely in logic elements—with a throughput of one output sample per clock cycle. With this architecture, there is a new output value every clock cycle.
- The serial CORDIC architecture uses fewer resources than the parallel CORDIC architecture. However, its throughput is reduced by a factor equal to the magnitude precision. For example, if you select a magnitude precision of *N* bits in the NCO MegaCore function, the output sample rate and the Nyquist frequency is reduced by a factor of *N*. This architecture is implemented entirely in logic elements and is useful if your design requires low frequency, high precision waveforms. With this architecture, the adder stages are stored internally and a new output value is produced every *N* clock cycles.

Refer to "Implementation Tab - CORDIC Algorithm" on page 3–4.

Multiplier-Based Architecture

The multiplier-based architecture uses multipliers to reduce memory usage. You can choose to implement the multipliers in either:

- Logic elements (Cyclone) or combinational ALUTs (Stratix).
- Dedicated multiplier circuitry (for example, dedicated DSP blocks) in device families that support this feature (Stratix III, Stratix II, Stratix GX, Stratix, or Arria GX devices).

When you specify a dual output multiplier-based NCO, the MegaCore function provides an option to output a sample every two clock cycles. This setting reduces the throughput by a factor of two and halves the resources required by the waveform generation unit. Refer to "Implementation Tab - Multiplier-Based Algorithm" on page 3–5.

Table 4–2 on page 4–8 summarizes the advantages of each algorithm.

Table 4–2. Architecture Comparison

Architecture	Advantages
Large ROM	Good for high speed and when a large quantity of internal memory is available. Gives the highest spectral purity and uses the fewest logic elements for a given parameterization.
Small ROM	Good for high output frequencies with reduced internal memory usage when a lower SFDR is acceptable.
CORDIC	High performance solution when internal memory is at a premium. The serial CORDIC architecture uses fewer resources than parallel although the throughput is reduced.
Multiplier-Based	Reduced memory usage by implementing multipliers in logic elements or dedicated circuitry.

Frequency Modulation

In the NCO MegaCore function, you can add an optional frequency modulator to your custom NCO variation. You can use the frequency modulator to vary the oscillator output frequency about a center frequency set by the input phase increment. This option is useful for applications in which the output frequency is tuned relative to a free-running frequency, for example in all-digital phase-lock-loops.

You can also use the frequency modulation input to switch the output frequency directly, for example, to implement frequency shift keying (FSK) modulators like the quaternary FSK modulator in "Example Design 2" on page A–3.

You can set the frequency modulation resolution input in the NCO MegaCore function. The specified value must be less than or equal to the phase accumulator precision. The NCO MegaCore function also provides an option to increase the modulator pipeline level; however, the effect of the increase on the performance of the NCO MegaCore function varies across NCO architectures and variations.

Phase Modulation

You can use the NCO MegaCore function to add an optional phase modulator to your MegaCore function variation, allowing dynamic phase shifting of the NCO output waveforms. This option is particularly useful if you want an initial phase offset in the output sinusoid. You can also use the option to implement efficient phase shift keying (PSK) modulators in which the input to the phase modulator varies according to a data stream. You set the resolution and pipeline level of the phase modulator in the NCO MegaCore function. The input resolution must be greater than or equal to the specified angular precision.

Phase Dithering

All digital sinusoidal synthesizers suffer from the effects of finite precision, which manifests itself as spurs in the spectral representation of the output sinusoid. Because of angular precision limitations, the derived phase of the oscillator tends to be periodic in time and contributes to the presence of spurious frequencies. You can reduce the noise at these frequencies by introducing a random signal of suitable variance into the derived phase, thereby reducing the likelihood of identical values over time. Adding noise into the data path raises the overall noise level within the oscillator, but tends to reduce the noise localization and can provide significant improvement in SFDR.

The extent to which you can reduce spur levels is dependent on many factors. The likelihood of repetition of derived phase values and resulting spurs, for a given angular precision, is closely linked to the ratio of the clock frequency to the desired output frequency. An integral ratio clearly results in high-level spurious frequencies, while an irrational relationship is less likely to result in highly correlated noise at harmonic frequencies.

The Altera NCO MegaCore function allows you to finely tune the variance of the dither sequence for your chosen algorithm, specified precision, and clock frequency to output frequency ratio, and dynamically view the effects on the output spectrum graphically. See "Example Design 1" on page A–1 for an example using phase dithering and its effect on the spectrum of the output signal.

Multi-Channel NCOs

The NCO MegaCore function allows you to implement multi-channel NCOs. This allows for multiple sinusoids of independent frequency and phase to be generated at a very low cost in additional resources. The resulting waveforms have an output sample-rate of f_{clk}/M where M is the number of channels. You can select 1 to 8 channels.

Multi-channel implementations are available for all single-cycle generation algorithms. The input phase increment, frequency modulation value and phase modulation input are input sequentially to the NCO with the input values corresponding to channel 0 first and channel (M–1) last. The inputs to channel 0 should be input on the rising clock edge immediately following the de-assertion of the NCO reset.

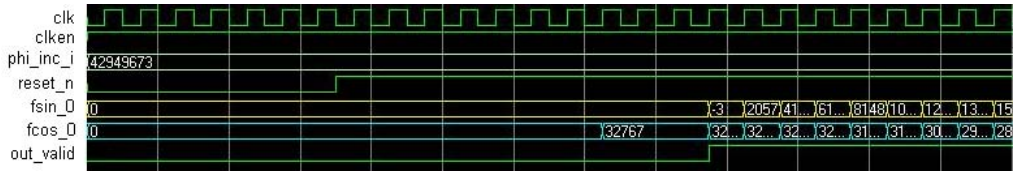
On the output side, the first output sample for channel 0 is output concurrent with the assertion of `out_valid` and the remaining outputs for channels 1 to (M–1) are output sequentially. See "Multi-Channel NCO Timing Diagram" on page 4–12 for details of how the data is provided to and received from a multi-channel NCO.

If a multi-channel implementation is selected, the NCO MegaCore function generates VHDL and Verilog test benches that time-division-multiplex the inputs into a single stream and de-multiplex the output streams into their respective down-sampled channelized outputs.

Timing Diagrams

Figure 4–4 shows the timing with a single clock cycle per output sample.

Figure 4–4. Single-Cycle Per Output Timing Diagram

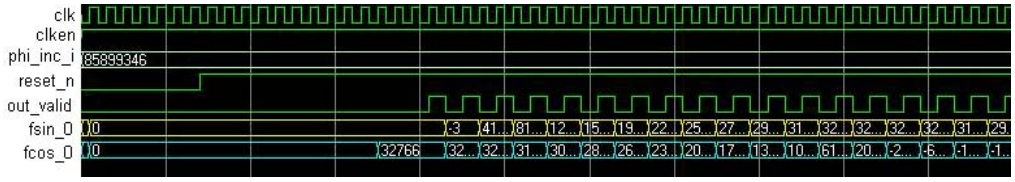


All NCO architectures—except for serial CORDIC and multi-cycle multiplier-based architectures—output a sample every clock cycle. After the clock enable is asserted, the oscillator outputs the sinusoidal samples at a rate of one sample per clock cycle, following an initial latency of L clock cycles. The exact value of L varies across architectures and parameterizations.

For the non-single-cycle per output architectures, the optional phase and frequency modulation inputs need to be valid at the same time as the corresponding phase increment value. The values should be sampled every 2 cycles for the two-cycle multiplier-based architecture and every N cycles for the serial CORDIC architecture, where N is the magnitude precision.

Figure 4–5 shows the timing diagram for a two-cycle multiplier-based NCO architecture.

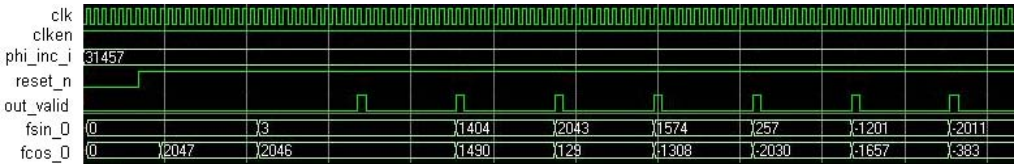
Figure 4–5. Two-Cycle Multiplier-Based Architecture Timing Diagram



After the clock enable is asserted, the oscillator outputs the sinusoidal samples at a rate of one sample for every two clock cycles, following an initial latency of *L* clock cycles. The exact value of *L* depends on the parameters that you set.

Figure 4–6 shows the timing diagram for a serial CORDIC NCO architecture.

Figure 4–6. Serial CORDIC Timing Diagram



Note that the `fsin_0` and `fcos_0` values can change while `out_valid` is low.

After the clock enable is asserted, the oscillator outputs sinusoidal samples at a rate of one sample per *N* clock cycles, where *N* is the magnitude precision set in the NCO MegaCore function. Figure 4–6 shows the case where *N* = 8. There is also an initial latency of *L* clock cycles; the exact value of *L* depends on the parameters that you set.

Table 4–3 shows typical latency values for the different architectures.

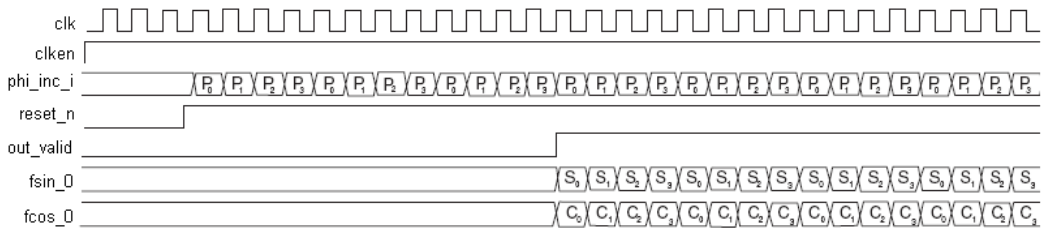
Table 4–3. Latency Values

Architecture	Variation	Latency (2), (3)		
		Base	Minimum	Maximum
Small ROM	all	7	7	13
Large ROM	all	4	4	10
Multiplier-Based	Throughput = 1, Logic cells	11	11	17
Multiplier-Based	Throughput = 1, Dedicated, Special case (1)	8	8	14
Multiplier-Based	Throughput = 1, Dedicated, Not special case	10	10	16
Multiplier-Based	Throughput = 1/2	15	15	26
CORDIC	Parallel	2N + 4	20 (4)	74 (5)
CORDIC	Serial CORDIC	2N + 2	18 (4)	258 (5)

Note for Table 4–3:
(1) Special case: (9 <= N <= 18 && WANT_SIN_AND_COS).
(2) Latency = base latency + dither latency+ frequency modulation pipeline + phase modulation pipeline (×N for serial CORDIC).
(3) Dither latency = 0 (dither disabled) or 2 (dither enabled).
(4) Minimum latency assumes N = 8.
(5) Maximum latency assumes N = 32

Figure 4–7 shows the timing diagram for a multi-channel NCO in the case where the number of channels, M is set to 4. The input phase increments for each channel, P_k are interleaved and loaded sequentially.

Figure 4–7. Multi-Channel NCO Timing Diagram



The phase increment for channel 0 is the first value read in on the rising edge of the clock following the de-assertion of `reset_n` (assuming `clken` is asserted) followed by the phase increments for the next (M-1) channels.

The output signal `out_valid` is asserted when the first valid sine and cosine outputs for channel 0, S_0, C_0, respectively are available. The output values S_k and C_k corresponding to channels 1 through (M-1) are output sequentially by the NCO. The outputs are interleaved so that a new output sample for channel k is available every M cycles.

Avalon Streaming Interface

The Avalon® Streaming (Avalon-ST) interface is an evolution of the Atlantic™ interface. The Avalon-ST interface defines a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface and simplifies the process of controlling the flow of data in a datapath.

Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. The NCO MegaCore function is an Avalon-ST source and does not support backpressure.

For more information on the Avalon-ST interface including integration with other Avalon-ST components which may support backpressure, refer to the *Avalon Interface Specification*.

Signals

The NCO MegaCore function supports the input and output signals shown in Table 4–4.

Table 4–4. NCO MegaCore FunctionSignals

Signal	Direction	Description
`clk`	Input	Clock.
`clken`	Input	Active high clock enable.
`freq_mod_i [F-1:0]`	Input	Optional frequency modulation input. You can specify the modulator resolution *F* in IP Toolbench.
`phase_mod_i [P-1:0]`	Input	Optional phase modulation input. You can specify the modulator precision *P* in IP Toolbench.
`phi_inc_i [A-1:0]`	Input	Input phase increment. You can specify the accumulator precision *A* in IP Toolbench.
`reset_n`	Input	Active-low reset.
`fcos_o [M-1:0]`	Output	Optional output cosine value (when dual output is selected). You can specify the magnitude precision *M* in IP Toolbench.

fsin_o [M-1:0]	Output	Output sine value. You can specify the magnitude precision *M* in IP Toolbench.
out_valid	Output	Data valid signal. Asserted by the MegaCore function when there is valid data to output.

References

Altera application notes, white papers, and user guides providing more detailed explanations of how to effectively design with MegaCore functions and the Quartus II software are available at the Altera web site (www.altera.com).

Refer also to the following references:

1. Andraka, Ray. *A Survey of CORDIC Algorithms for FPGAs,* FPGA '98 Proceedings of the ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays.

Appendix A. Example Designs

Example Design 1

Example design 1 is a high-precision, dual-output oscillator for use in an intermediate frequency (IF) I-Q modulator. The design targets the Altera® EP2S15F484C3 Stratix® II device.

The top-level design file is:

*<install path>***\ip\nco\example_designs\design1\design1.bdf**

The oscillator meets the following specifications:

- SFDR: 110 dB
- Output Sample Rate: 300 MSPS
- Output Frequency: 21 MHz
- Frequency Resolution: 0.07 Hz

To meet these requirements, the design uses the following NCO MegaCore function settings:

- *Multiplier-based algorithm*—By using the dedicated multiplier circuitry in Stratix devices, the NCO architectures that implement this algorithm can provide very high performance.
- *Clock rate of 300 MHz and 32-bit phase accumulator precision*—These settings yield a frequency resolution of 70 mHz.
- *Angular and magnitude precision*—These settings are critical to meet the SFDR requirement, while minimizing the required device resources. Setting the angular precision to 17 bits and the magnitude precision to 18 bits results in the spectrum shown in Figure A–1 on page A–2.
- *Dither level*—The angular and magnitude precision settings described above yield an SFDR of approximately 102.8 dB, which is clearly not sufficient to meet the specification. Using the dither control in the NCO MegaCore function, the variance of the dithering sequence is increased until the trade-off point between spur reduction and noise level augmentation is reached for these particular clock-frequency to output frequency ratio and precision settings as shown in Figure A–2 on page A–2. At a dithering level of 5, the SFDR is approximately 111.95 dB, which exceeds the specification.

Figure A–1. Spectrum After Setting Angular & Magnitude Precision

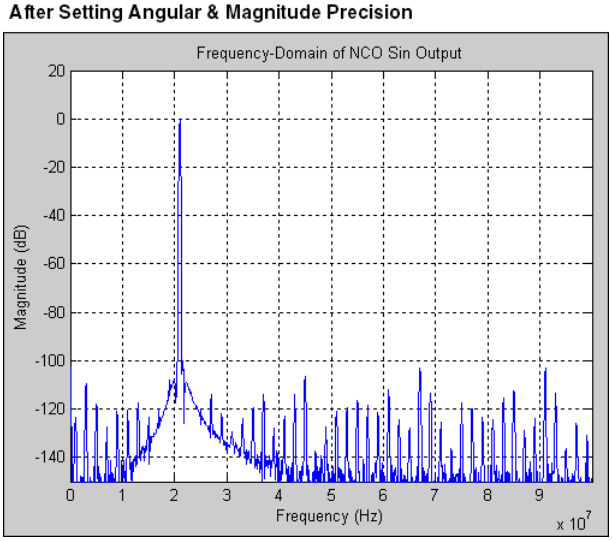
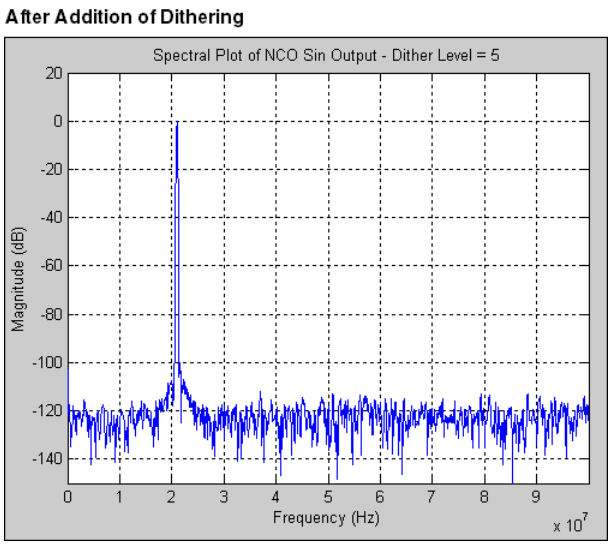


Figure A–2. Spectrum After the Addition of Dithering



Example Design 2

Example design 2 is a quaternary frequency shift keying (QFSK) modulator for use in a hypothetical transmitter design. The design targets the Altera EP2S15F484C3 Stratix II device. In this type of modulator, the output frequency of the oscillator varies according to an input symbol stream, the values of which map to a four-symbol alphabet.

The top-level design file is:

<*install path*>**\ip\nco example_designs\design2\design2.bdf**

The oscillator meets the following specifications:

- SFDR: 80 dB
- Output Frequencies:
  - f_c - 5.76 MHz
  - f_c - 1.92 MHz
  - f_c + 1.92 MHz
  - f_c + 5.76 MHz
- Output Sample Rate: 220 MSPS
- Frequency Resolution: 0.05 Hz

Where f_c is the free-running output frequency of the NCO.

Table A–1 shows the mapping of the symbols to output frequencies, assuming that the free-running frequency is set to 15.36 MHz by setting the phase increment of the oscillator to 299866808.

Table A–1. Symbol Mapping

Binary Symbol	Frequency Modulation Value	Output Frequency (MHz)
10	4182517243	9.60
11	4257483945	13.44
01	37483351	17.28
00	112450053	21.12

To meet these requirements, the design uses the following NCO MegaCore function settings:

- *Parallel CORDIC algorithm*—In the overall hypothetical design, other system blocks use much of the Stratix device's internal memory and DSP blocks; however, logic elements (LEs) are in abundant supply. The need for a high precision, high performance oscillator that uses logic elements only makes the Parallel CORDIC algorithm a suitable choice.

- *Phase accumulator precision*—The frequency resolution specifications demand a phase accumulator precision of 32 bits.
- *Frequency modulator precision*—To maximize the frequency resolution of the modulating signal, the resolution of the frequency modulator is also set to 32 bits of precision.
- *Angular and magnitude precision and dithering*—An angular precision of 15 bits, a magnitude precision of 14 bits, and a dithering level of 5 meet the requirements of 80 dB SFDR for the four possible output frequencies as shown in Figure A–3.

Figure A–3. Output Frequencies

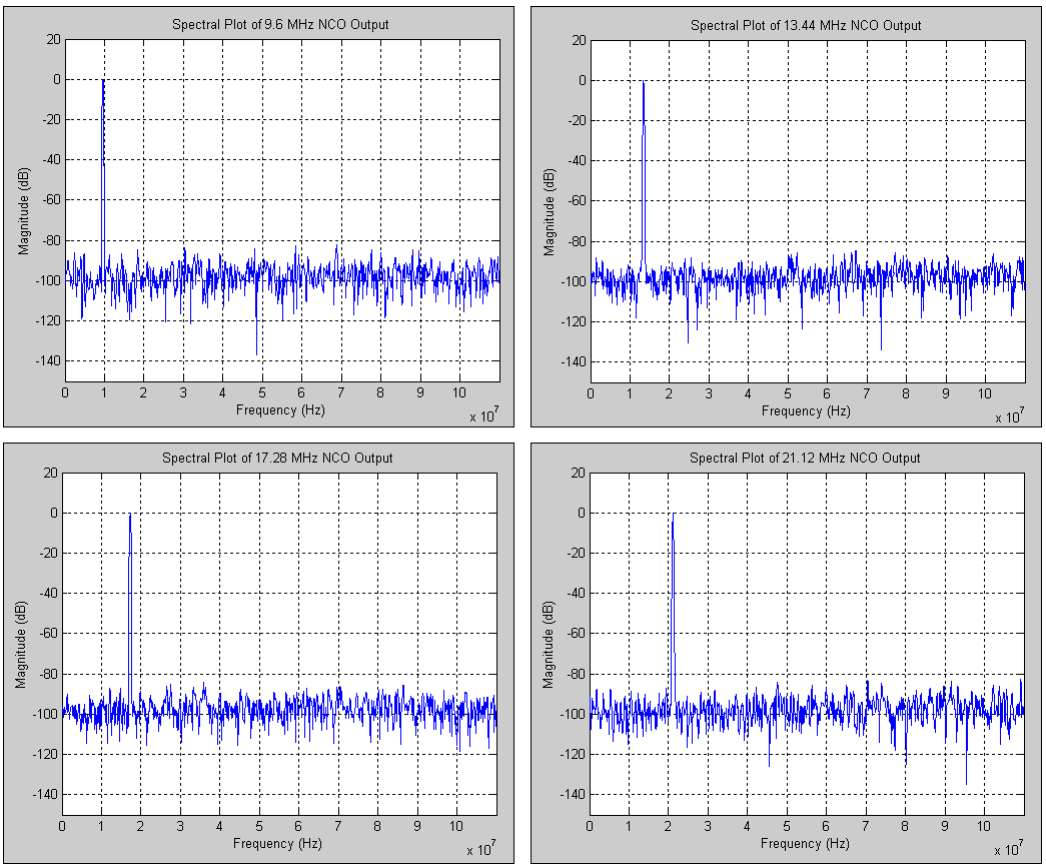
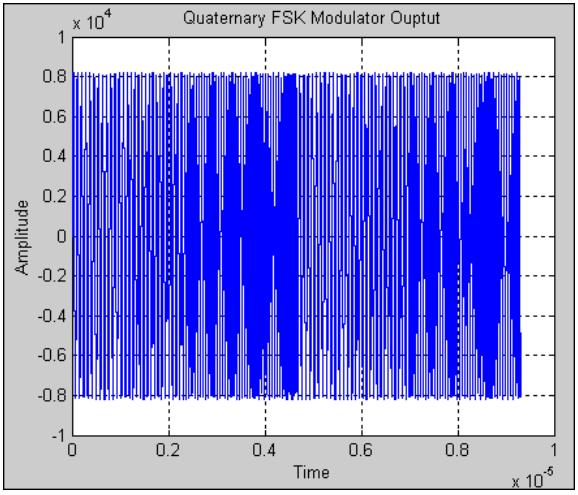

Spectral Plot of 9.6 MHz NCO Output
Spectral Plot of 13.44 MHz NCO Output
Spectral Plot of 17.28 MHz NCO Output
Spectral Plot of 21.12 MHz NCO Output
Magnitude (dB)
Frequency (Hz)
x 10^7

Figure A–4 shows a segment of the resulting FSK modulated waveform in the time-domain.

Figure A–4. FSK Modulated Waveform



The generated output frequency for a given phase increment, f_0 and frequency modulation increment, f_{FM} can be calculated using Equations 3 and 6 on page 4–4.

The following parameters are specified in the **nco_design_tb.m** file:

- ϕ_{INC} = 299866808
- ϕ_{FM} = 268435456
- f_{clk} = 220MHz
- M = 32
- F = 32

This parameterization gives the following values for f_0 and f_{FM}:

- f_0 = 15.36MHz
- f_{FM} =13.76MHz

Additional Information

Revision History

The following table displays the revision history for this user guide.

Date	Version	Changes Made
May 2008	8.0	• Separated the design flows and parameter setting sections. • Full support for Cyclone® III • Preliminary support for Stratix® IV
October 2007	7.2	• Updated NCO block diagram • Added multi-channel description and timing diagram • Added latency table • Updated GUI screenshots • Full support for Arria™ GX
May 2007	7.1	• Added 32-bit precision for angle & magnitude • Preliminary support for Arria™ GX • Full support for Stratix II GX and HardCopy II devices.
December 2006	7.0	• Preliminary support for Cyclone III
December 2006	6.1	• Preliminary support for Stratix III • Minor updates throughout the user guide
April 2006	2.3.1	• Maintenance release; updated screen shots and format
October 2005	2.3.0	• Maintenance release; updated screen shots and format • Preliminary support for HardCopy® II, HardCopy Stratix, and Stratix II GX • Removed Mercury and Excalibur device support
June 2004	2.2.0	• Added Cyclone II support • Updated functional description, tutorial instructions and screenshots
February 2004	2.1.0	• Enhancements include support for Stratix II devices; support for easy-to-use IP Toolbench; IP functional simulation models for use in Altera®-supported VHDL and Verilog HDL simulators; support for UNIX and Linux operating systems.
November 2002	2.0.2	• Updated the screen shots; made some formatting and organization changes; minor wording changes to several sections.
July 2002	2.0.1	• NCO MegaCore functions now display a single DSP Builder library for OpenCore and OpenCore Plus in the Simulink Library Browser.
May 2002	2.0.0	• Updated functional description. Added DSP Builder, OpenCore Plus, and licensing information. Removed reference designs and replaced with example designs. Updated all screen shots. Made formatting and organization changes.
April 2000	1.0	• Version 1.0 of this user guide.

How to Contact Altera

For the most up-to-date information about Altera® products, refer to the following table.

Contact *(1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note to table:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown in the following table.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions."

Visual Cue	Meaning
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (for example, the VHDL keyword `BEGIN`), as well as logic function names (for example, `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	The caution calls attention to a condition that could damage the product or design and should be read prior to starting or continuing with the procedure or process.
	The warning calls attention to a condition that could cause injury to the user and should be read prior to starting or continuing the procedure or processes.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.